10 集成学习笔记

10.1 集成学习概述

10.1.1 集成学习的定义及原理

- **集成学习**：将多个弱学习器组合成一个强学习器，提高模型准确性。
- **个体学习器**：集成学习的基本单元，可以是不同类型的学习算法。
- **Bagging**：通过有放回地采样训练多个弱学习器，然后平均或投票预测结果。
- **Boosting**：迭代训练一系列弱学习器，每个学习器都试图纠正前一个学习器的错误。
- **Stacking**：训练多个弱学习器，然后将它们的预测结果作为新的特征输入到另一个学习器中。

常用的库有如下几种：

库	主要用途
Scikit-learn	传统机器学习模型与集成学习
XGBoost	梯度提升决策树（GBDT）
LightGBM	轻量级梯度提升模型，适用于大规模数据
CatBoost	处理类别特征更高效的 GBDT

10.1.2 Bagging（袋装）

Bagging通过构建多个独立的决策树，平均或投票预测结果。通过特征随机采样（Random Subspaces）或样本随机采样（Random Patches），增加模型多样性，减少过拟合。

10.1.3 Boosting（提升）

Boosting通过迭代训练一系列弱学习器，每个学习器都试图纠正前一个学习器的错误。通过调整样本权重，使模型更关注难以分类的样本。

10.1.4 Stacking（堆叠）

Stacking通过训练多个弱学习器，然后将它们的预测结果作为新的特征输入到另一个学习器中。这种方法可以整合多个模型的预测结果，提高整体预测的准确性和鲁棒性。

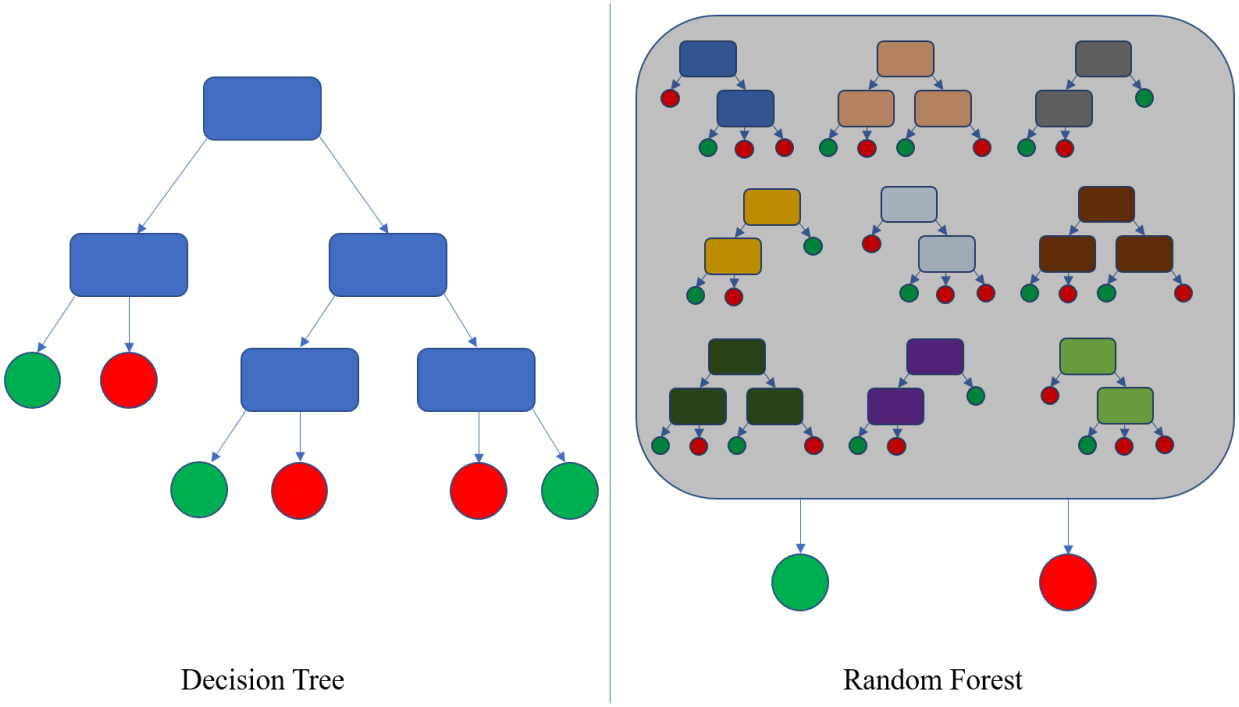
10.2 随机森林

10.2.1 随机森林基本原理

随机森林（Random Forest）是一种基于决策树的集成学习算法，它通过构建多个决策树并结合它们的预测结果来提高模型的预测准确性和鲁棒性。随机森林通过引入特征随机采样和样本随机采样（Bootstrap sampling）来减少过拟合，增加模型的泛化能力。

- **特征随机采样**：在构建每棵树时，随机选择一部分特征作为分裂节点的候选特征。
- **样本随机采样**：通过有放回抽样（Bootstrap sampling）的方式从原始数据集中抽取多个子样本集，每个子样本集用于构建一棵树。

随机森林的预测结果通常是通过多数投票（分类问题）或平均值（回归问题）来决定的。



Decision Tree　　Random Forest

10.2.2 随机森林算法应用与评价

随机森林算法因其简单、高效和良好的性能，被广泛应用于各种分类和回归任务中。它在处理高维数据、缺失数据和不平衡数据集时表现出色，且对于特征选择和参数调整不敏感。

- **优点**:
  - 易于并行化，适合大规模数据集。
  - 对于特征选择不敏感，可以处理大量特征。
  - 具有很好的鲁棒性和准确性。
  - 可以自然地处理缺失数据。
- **缺点**:
  - 模型可解释性较差，因为模型由多棵树组成。
  - 在某些特定情况下可能过拟合，特别是当树的深度较大时。

10.2.3 实践案例：基于随机森林算法的银行危机案例

在银行危机预测案例中，我们使用随机森林算法来预测银行是否会面临危机。数据集包含多个特征，如资产规模、负债率、盈利能力等，以及目标变量（是否发生危机）。

数据预处理

首先，需要对数据进行预处理，包括处理缺失值、编码分类变量等。

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# 加载数据
data = pd.read_csv('bank_crisis_data.csv')

# 数据预处理
# 处理缺失值、编码分类变量等
# ...

# 划分训练集和测试集
X = data.drop('target', axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

接下来，使用随机森林算法训练模型。

```
# 创建随机森林模型
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)

# 训练模型
rf_clf.fit(X_train, y_train)
```

最后，评估模型的性能。

```
# 预测
y_pred = rf_clf.predict(X_test)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

通过上述步骤，可以使用随机森林算法有效地预测银行危机，并对模型的性能进行评估。

10.3 AdaBoost

10.3.1 加法模型

- AdaBoost通过调整样本权重，使模型更关注难以分类的样本。
- 加法模型表示为多个弱学习器的加权和。

10.3.2 前向分布算法

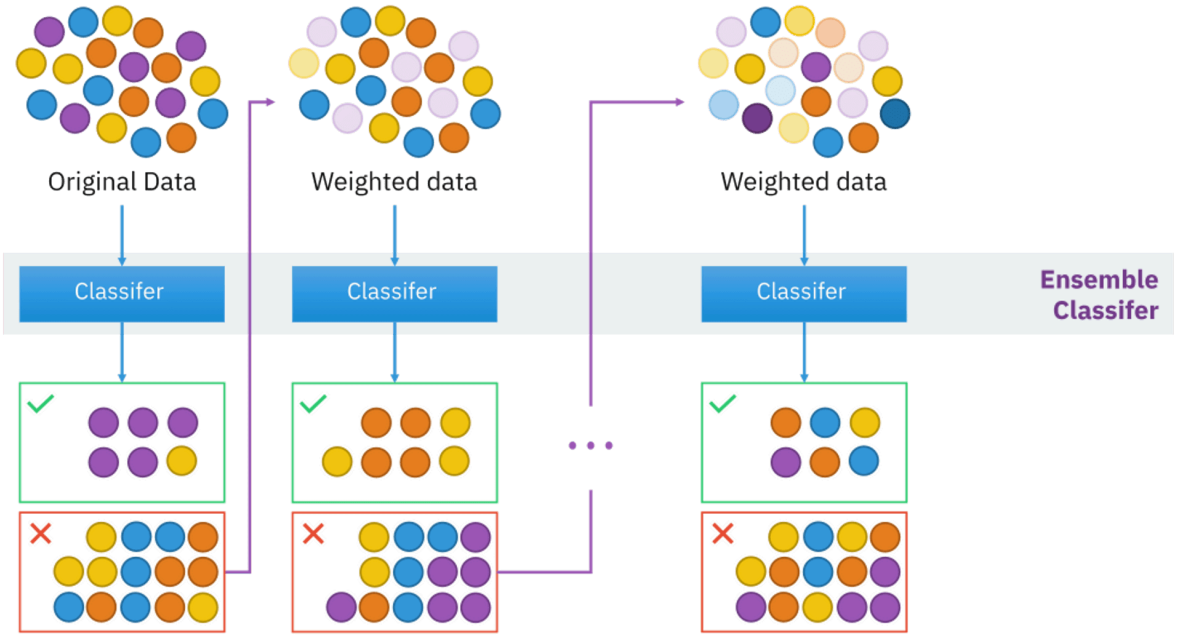
- 前向分布算法是一种贪心算法，用于构建AdaBoost模型。
- 逐步增加被错误分类样本的权重，降低正确分类样本的权重。

10.3.3 AdaBoost 求解步骤

1. 初始化样本权重。
2. 训练弱学习器并计算分类误差。
3. 更新样本权重，增加错误分类样本的权重。
4. 重复上述步骤直到达到预定迭代次数或满足停止条件。

10.3.4 AdaBoost 算法应用与评价

- 通过动态调整样本权重，聚焦先前分类错误的样本，提高整体性能。
- 适用于处理复杂数据集，尤其在数据分布不均匀时表现良好。



10.3.5 实践案例：基于 AdaBoost 的马疝病预测

- 使用Python中的`sklearn.ensemble`模块实现AdaBoost。
- 多次重复实验确认结果稳定性，并与决策树和GBDT对比。
  代码如下：

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 加载数据集
iris = load_iris()
X, y = iris.data, iris.target

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 创建AdaBoost模型
ada_clf = AdaBoostClassifier(n_estimators=100, random_state=42)
ada_clf.fit(X_train, y_train)

# 预测
y_pred = ada_clf.predict(X_test)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

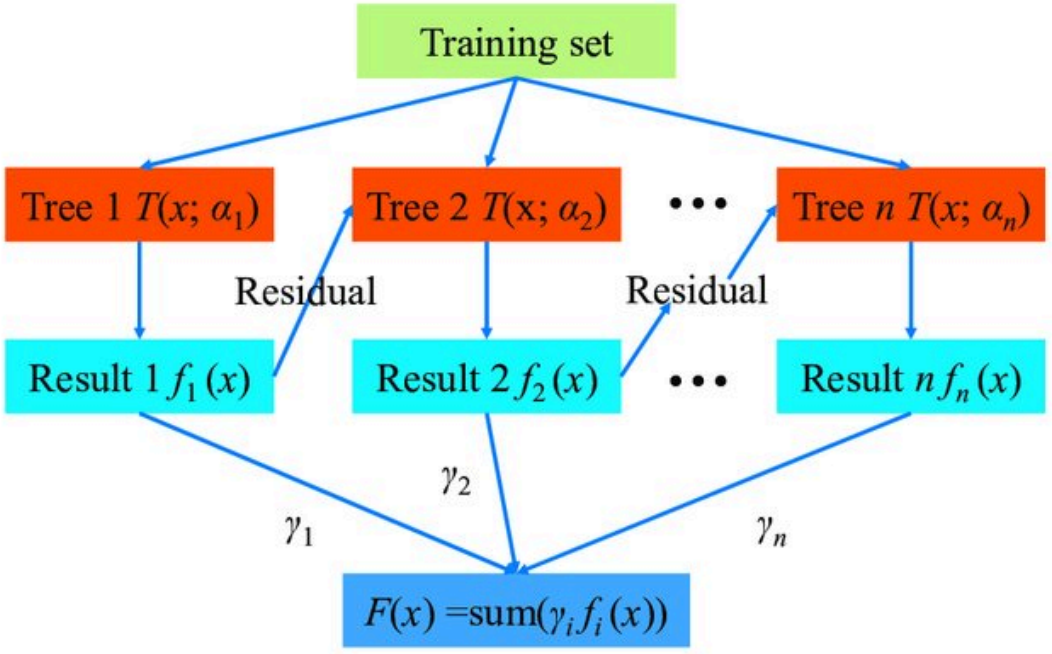
10.4 梯度提升树

10.4.1 回归树基本原理

- 回归树通过递归地将特征空间划分为多个子区域，并在每个子区域内拟合回归模型。
- 最小化残差平方和（RSS）选择最优划分特征和点。

10.4.2 梯度提升树基本原理

- GBDT通过迭代训练决策树拟合损失函数的负梯度。
- 逐步减少训练误差，提升模型性能。



10.4.3 实践案例：基于梯度提升树算法的充电桩故障状态预测

- 使用Python中的`sklearn.ensemble`模块实现GBDT。
- 对新能源汽车充电桩的故障状态进行分类预测，并与决策树和AdaBoost进行性能比较。
  代码如下：

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 加载数据集
iris = load_iris()
X, y = iris.data, iris.target

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 创建GBDT模型
gbdt_clf = GradientBoostingClassifier(n_estimators=100, random_state=42)
gbdt_clf.fit(X_train, y_train)

# 预测
y_pred = gbdt_clf.predict(X_test)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

10.4.4 GBDT 算法应用与评价

- GBDT通过迭代训练决策树，逼近训练数据的真实分布，提高泛化能力。
- 处理高维稀疏数据时可能效率较差，尤其在文本特征上表现较弱。

10.5 XGBoost

10.5.1 XGBoost 基本原理

- XGBoost是一种高效的梯度提升树算法，引入正则化项、列采样等技术，加速训练过程。
- 通过并行化处理、正则化技术和列采样等方法，克服GBDT的并行性和效率限制。

10.5.2 XGBoost 目标函数构建

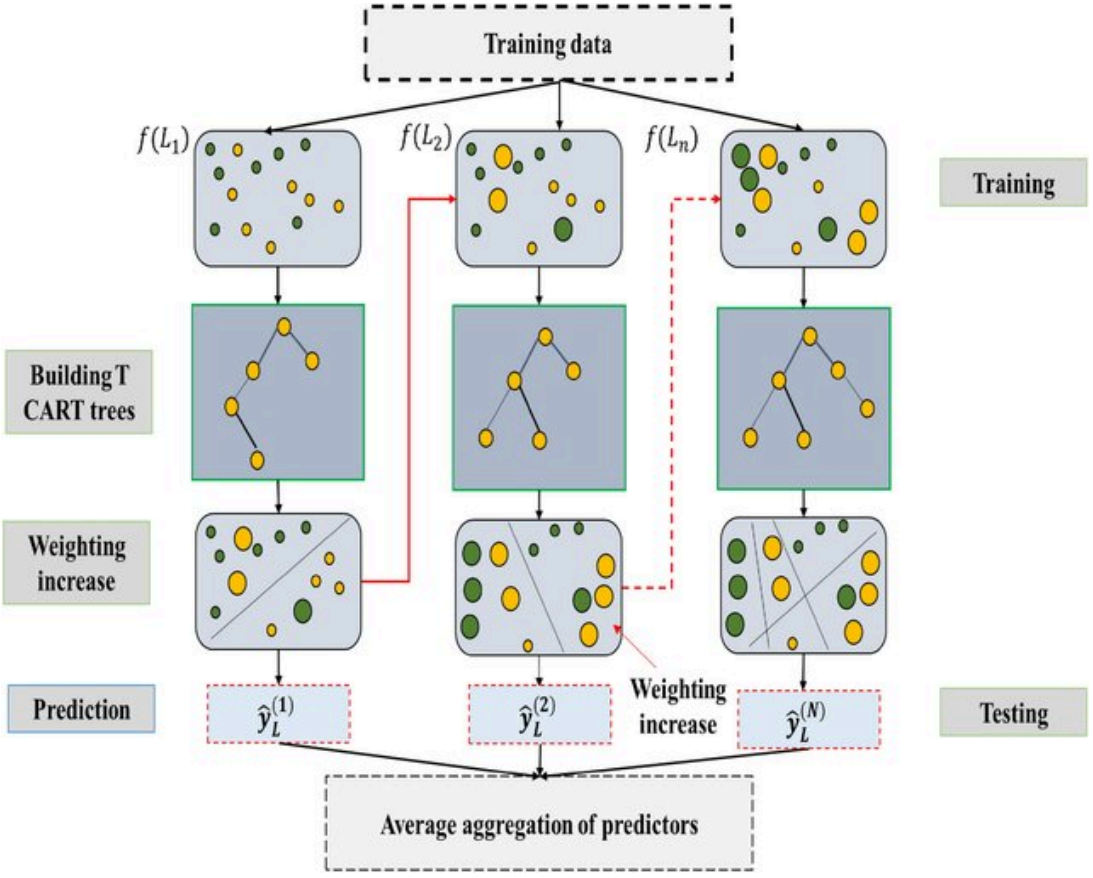
- XGBoost采用加法模型训练方法，通过损失函数和正则化项的组合优化模型。
- 目标函数由训练误差和正则化项组成，损失函数衡量预测误差，正则化项控制模型复杂度。

10.5.3 XGBoost 目标函数求解

- 使用二阶泰勒展开近似损失函数，有效优化目标函数并加速模型训练。
- 通过前向分布算法进行贪婪求解，逐步构建每一轮的弱学习器。

10.5.4 XGBoost 算法应用与评价

- 通过对损失函数进行二阶导数近似，自由定义和应用各种损失函数。
- 在正则化方面表现出色，支持L1和L2正则化，有效防止模型过拟合。



10.5.5 实践案例：基于 XGBoost 算法的产品定价预测

- 使用XGBoost算法对产品定价进行预测，数据集包含手机产品的型号、配置信息及价格范围。
- 调整XGBoost模型的超参数（如max_depth、learning_rate等），观察其对准确率的影响。
  代码如下：

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 加载数据集
data = pd.read_csv('data/cellphone.csv')
X = data.drop('price_range', axis=1)
y = data['price_range']
```

```
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 创建XGBoost模型
xgb_clf = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_clf.fit(X_train, y_train)

# 预测
y_pred = xgb_clf.predict(X_test)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

10.6 LightGBM

10.6.1 LightGBM 基本原理

- LightGBM是一种高性能梯度提升框架，采用高效的并行计算和直方图算法。
- 在训练和预测速度方面取得显著突破，适用于大规模数据集。

10.6.2 LightGBM 直方图算法

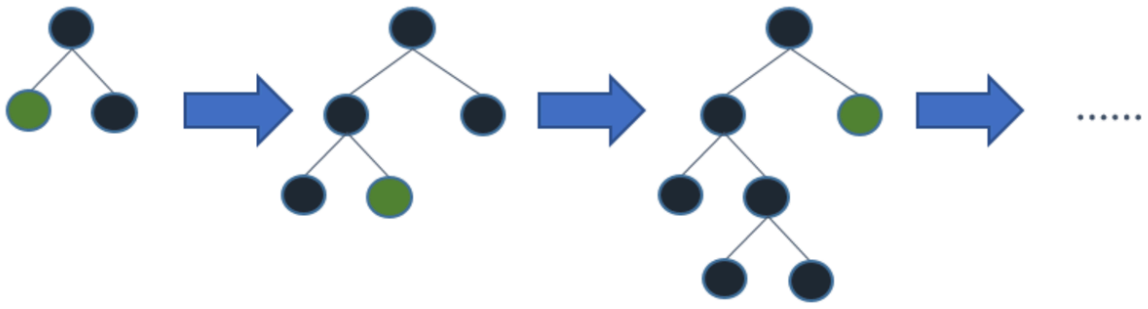
- LightGBM引入直方图算法（HA），通过对特征值进行分箱加速决策树构建。
- HA算法通过减少计算复杂度和列采样提高模型训练效率。

10.6.3 LightGBM 互斥特征捆绑算法

- 互斥特征捆绑算法（EFB）通过将高度相关的特征捆绑在一起，减少特征数量，降低模型复杂度。
- EFB算法在不影响模型精度的前提下，有效减轻计算负担。

10.6.4 LightGBM 算法应用与评价

- LightGBM在处理大规模数据集和高维特征数据集时表现出色，适用于各种实际场景。
- 需要存储额外的直方图和梯度信息，因此在内存消耗较大时可能成为限制因素。



10.6.5 实践案例：基于 LightGBM 算法的中风预测

- 使用LightGBM算法对中风症状进行预测，数据集包含患者的年龄、性别、高血压等信息。
- 调整LightGBM模型的超参数（如boosting_type、num_leaves等），观察其对准确率的影响。
  代码如下：

```
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 加载数据集
data = pd.read_csv('data/stroke.csv')
```

```
X = data.drop('stroke', axis=1)
y = data['stroke']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 创建LightGBM模型
lgbm = lgb.LGBMClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
lgbm.fit(X_train, y_train)

# 预测
y_pred = lgbm.predict(X_test)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

10.7 集成学习总结

- 集成学习通过组合多个弱学习器提升模型性能，适用于各种实际场景。
- 常见集成学习方法包括Bagging、Boosting、Stacking等，各有优缺点和适用场景。
- 通过调整模型的超参数和选择合适的集成方法，可以显著提升模型的预测性能和鲁棒性。

下面是一个集成学习实际案例，使用XGBoost实现多分类预测的实践：

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import seaborn as sns
import gc

## load data
train_data = pd.read_csv('../../data/train.csv')
test_data = pd.read_csv('../../data/test.csv')
num_round = 1000

## category feature one_hot
test_data['label'] = -1
data = pd.concat([train_data, test_data])
cate_feature = ['gender', 'cell_province', 'id_province', 'id_city', 'rate', 'term']
for item in cate_feature:
    data[item] = LabelEncoder().fit_transform(data[item])
    item_dummies = pd.get_dummies(data[item])
    item_dummies.columns = [item + str(i + 1) for i in range(item_dummies.shape[1])]
    data = pd.concat([data, item_dummies], axis=1)
data.drop(cate_feature,axis=1,inplace=True)

train = data[data['label'] != -1]
test = data[data['label'] == -1]

##Clean up the memory
del data, train_data, test_data
gc.collect()

## get train feature
del_feature = ['auditing_date', 'due_date', 'label']
features = [i for i in train.columns if i not in del_feature]

## Convert the label to two categories
train_x = train[features]
train_y = train['label'].astype(int).values
```

```
test = test[features]

params = {
    'booster': 'gbtree',
    'objective': 'multi:softmax',
    # 'objective': 'multi:softprob',   #Multiclassification probability
    'num_class': 33,
    'eval_metric': 'mlogloss',
    'gamma': 0.1,
    'max_depth': 8,
    'alpha': 0,
    'lambda': 0,
    'subsample': 0.7,
    'colsample_bytree': 0.5,
    'min_child_weight': 3,
    'silent': 0,
    'eta': 0.03,
    'nthread': -1,
    'missing': 1,
    'seed': 2019,
}

folds = KFold(n_splits=5, shuffle=True, random_state=2019)
prob_oof = np.zeros(train_x.shape[0])
test_pred_prob = np.zeros(test.shape[0])


## train and predict
feature_importance_df = pd.DataFrame()
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train)):
    print("fold {}".format(fold_ + 1))
    trn_data = xgb.DMatrix(train_x.iloc[trn_idx], label=train_y[trn_idx])
    val_data = xgb.DMatrix(train_x.iloc[val_idx], label=train_y[val_idx])

    watchlist = [(trn_data, 'train'), (val_data, 'valid')]
    clf = xgb.train(params, trn_data, num_round, watchlist, verbose_eval=20,
early_stopping_rounds=50)

    prob_oof[val_idx] = clf.predict(xgb.DMatrix(train_x.iloc[val_idx]),
ntree_limit=clf.best_ntree_limit)
    fold_importance_df = pd.DataFrame()
    fold_importance_df["Feature"] = clf.get_fscore().keys()
    fold_importance_df["importance"] = clf.get_fscore().values()
    fold_importance_df["fold"] = fold_ + 1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

    test_pred_prob += clf.predict(xgb.DMatrix(test), ntree_limit=clf.best_ntree_limit) /
folds.n_splits
result = np.argmax(test_pred_prob, axis=1)


## plot feature importance
cols = (feature_importance_df[["Feature",
"importance"]].groupby("Feature").mean().sort_values(by="importance", ascending=False).index)
best_features =
feature_importance_df.loc[feature_importance_df.Feature.isin(cols)].sort_values(by='importance',asc
ending=False)
plt.figure(figsize=(8, 15))
sns.barplot(y="Feature",
            x="importance",
            data=best_features.sort_values(by="importance", ascending=False))
plt.title('LightGBM Features (avg over folds)')
plt.tight_layout()
plt.savefig('../../result/xgb_importances.png')
```

10.7.1 集成学习在建筑工程中的应用

- 质量检测与缺陷预测
  - **随机森林** 可用于分析建筑材料的质量特征，并预测缺陷发生的可能性。
  - **XGBoost** 在裂缝检测、材料强度预测等问题上表现优秀。
- 施工安全管理
  - **Adaboost 和 Gradient Boosting** 可用于施工事故预测，结合历史数据识别潜在风险。
  - 结合**传感器数据**，通过集成学习优化安全监测系统。
- 能耗预测与优化
  - **Stacking 模型** 结合多个回归模型（如线性回归、XGBoost、LSTM），提高建筑能耗预测精度。
  - **随机森林 + 神经网络** 组合可用于优化建筑能源管理系统。
- 结构健康监测（SHM）
  - 采用 **Bagging（随机森林）** 识别结构损伤模式，提高检测稳定性。
  - 结合 **Boosting 方法（XGBoost）** 提高损伤预测的准确度。

10.7.2 未来趋势

- **结合深度学习**：集成 CNN、RNN 等模型，提升在图像、时序数据中的应用。
- **自适应集成学习**：动态调整基学习器的权重，提高模型适应性。
- **自动化机器学习（AutoML）**：自动优化集成策略，提高模型选择与参数调整的效率。